UAV Collision Avoidance using A* Algorithm

by

Tingsheng Liao

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 7, 2012

Keywords: UAV, collision avoidance, A* algorithm, heuristic

Approved by

Abstract

Collision avoidance is the essential requirement for unmanned aerial vehicles (UAVs) to become fully autonomous. Several algorithms have been proposed to do the path planning in a simulated environment, but only few can make them effectively survive in a dynamic environment. This issue keeps UAVs from commercial and other applications because when the UAVs fly autonomously, the inability to reliably sense and avoid other aircraft in the air can cause serious hazards.

In this thesis, we review several approaches including $A*$ algorithm, total field sensing approach, and Markov decision process. Then, a modification of $A*$ algorithm is proposed. Typically, $A*$ algorithm is implemented in a mobile robot system for the path planning in a static environment. We introduce some approaches to allow us using $A*$ algorithm in a dynamic environment. The evaluation of this algorithm is based on the simulation of different scenarios, and the comparison between two heuristic functions will be detailed. We discuss the performance of our approach, the suitable condition for it to work reliably, and what issues could affect its performance. We also investigate the limitation of our approach in the extreme scenarios to provide useful suggestions for improvement.

Table of Contents

List of Figures

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) are aircraft with no human pilot on board. They are either remotely controlled by human pilots or pre-programmed to fly autonomously. A UAV could be a fixed-wing airplane, a helicopter, a robotic bird or any vehicle moving in the air. Initially, UAVs were designed and used for military purposes such as reconnaissance and attack missions. In recent conflicts, UAVs executed numerous spying and combat missions. Their growing capabilities and successes are drawing increasing attention for potential commercial applications in agriculture, land surveying, and oil pipeline inspection among many others.

However, there is one issue which keeps UAVs from spread commercial deployment: the Federal Aviation Administration **(FAA)** strictly requires that for UAVs to be integrated into the national airspace, they must be at least as competent (capable of operating safely) as an equivalent human pilot *without* cooperative communication (such as commands from a human controller or information from neighboring aircraft) [15]. When UAVs fly autonomously, their inability to reliably sense and avoid other aircraft are a main concern and exclude them from commercial applications. UAVs must avoid collisions.

There are numerous existing approaches to solve the UAV collision avoidance problem. For example, Park [32] applied the point of closest approach method to achieve UAV conflicts detection and collision resolution. Some research groups used grid-based approaches to achieve collision avoidance: Tooren [40] adapted $A*$ algorithm to avoid dynamic obstacles. The $A*$ (pronounced "A star") algorithm is a grid-based approach, and it was originally designed for terrestrial robots that can speed up, slow down, stop, sharply turn, or even back up along **fixed** obstacles. In this study, we will adapt the A^* algorithm for UAVs path planning so that it is capable of avoiding other UAVs.

In this work, we assume that the UAVs fly at a constant speed, do not change altitude, and turn by 22^o per second at most. Constant speed and altitude optimize energy efficiency and stealth. Although the $A*$ algorithm is frequently used to solve the UAVs path planning, only a few researchers apply it to achieve collision avoidance. Those previous works using the $A*$ algorithm for collision avoidance did not discuss much about the relation between heuristics and environmental parameters (field size, number of UAVs, etc.). We wonder: *How does the performance differ by manipulating the heuristic function of the A^* algorithm and environmental conditions in the UAVs collision avoidance problem?*

In this study, we want to answer this question by performing simulations with several scenarios. All of the data are simulated under C++ and Matlab programming environment. As a result, we are able to determine the limitations for the $A*$ algorithm using different heuristic functions. For example, we will show how far the UAVs have to deviate in order to keep in the safety path, and how many UAVs in a certain field will cause unsolvable collision.

In this thesis, we first review the related works with different approaches. Our model will then be proposed to demonstrate how do we solve the dynamic collision avoidance problem. Next, we will perform several simulations in different scenarios to evaluate our approach. Finally, we propose our conclusion and suggestion for the future work.

Chapter 2

Literature Review

Numerous approaches have been proposed to address collisions avoidance for UAVs. We review the geometric, stochastic, linear programming , artificial potential fields(**APF**), and grid-based approaches. For each approach, we review the representative method.

2.1 Geometric Approach

In this approach, each UAV is considered as a point mass, its future path can be predicted based on its most recent trajectory, current speed and bearing. A representative algorithm is the *Point of Closest* approach (**PCA**)[32]. Given the predicted paths of all UAVs, *PCA* method computes the mutual distances between UAVs. Should the distance be smaller than some *safety* threshold distance, evading strategies are enacted to avoid the collisions.

2.1.1 Free Flight

In 1991, International Civil Aviation Organization created the Future Air Navigation System Panel. The concept of "user define trajectory" was proposed and became known as "free flight" [24] [4] by the mid-1990s. The concept of free flight is to replace the current air traffic management methods by the use of growing technology nowadays. The ultimate goal of free flight is not to use centralized control (such as ground station), but automatically fly in a distributed way using computer communication to ensure the Alert Zone of air flights will not be violated. Once the Alert Zone is violated, air traffic controllers will intervene to assist in conflict avoidance to prevent the violation of Protected Airspace Zone. Fig. 2.1

illustrates the Alert Zone and Protected Airspace Zone. This concept motivated numerous research group to develop autonomous conflict detection and resolution methods. In 1997, Krozel and Peters proposed an economic model for conflict detection and resolution using *PCA* method. [24]

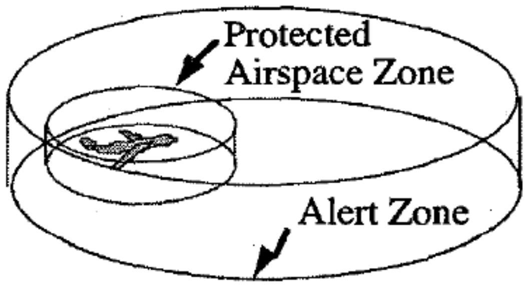


Fig. 2.1: Illustration of free flight protected and alert zones. [24]

2.1.2 Point of Closest Approach

Krozel and Peters [24] considered two UAVs flying without sideslip which are going to have an encounter with each other as shown in Fig. 2.2. In order to predict whether a collision will happen or not, they use geometry method to calculate the minimum distance of these two UAVs passing by each other. In other word, they use *PCA* method to get the miss distance.

The miss distance vector $\vec{r}$ is defined:

$$\vec{r}_m = \hat{c} \times (\vec{r} \times \hat{c}) \tag{2.1}$$

where $\vec{r}$ is the relative distance vector of UAV'B' with respect to UAV'A', and $\hat{c}$ is the unit vector in the direction of the relative velocity vector $\vec{c}$:

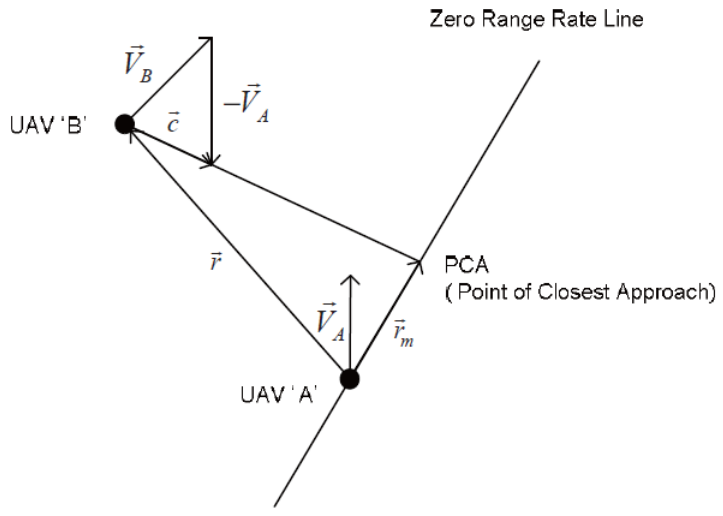


Fig. 2.2: Illustration for Point of Closest Approach method. [32]

$$\vec{c} = \vec{V}_B - \vec{V}_A \tag{2.2}$$

By the definition of miss distance vector $\vec{r}$, we can naturally know that the miss vector $\vec{r}$ and the relative velocity vector $\vec{c}$ are orthogonal:

$$\vec{r}_m \cdot \vec{c} = 0 \tag{2.3}$$

The relative motion from UAV'B' to UAV'A' will follow along the direction of $\vec{c}$, and after the time of closest approach τ, the relative motion will be located at the vector location $\vec{r}_m$:

$$\vec{r}_m = \vec{r} + \vec{c}\tau \tag{2.4}$$

with Eq.2.3 and 2.4, we can calculate the time of closest approach τ:

$$\tau = -\frac{\tau \cdot \vec{c}}{\vec{c} \cdot \vec{c}} \tag{2.5}$$

In 1958, Morrel [30] used simply range divided by range rate as the predicted time-to-collision criteria. The time of closest approach τ differs from Morrel's criteria, the time of closest approach is more reliable and effective to judge weather a collision will occur or not. In the example above, the altitude is fixed so that the UAVs can only fly in a horizontal plane. Krozel and Peters' also described the full 3D conflict detection analysis was described in [23].

Park [32] indicated that from Eq. 2.5, the time of closest approach τ will be a positive value when two of UAVs are getting closer, and τ will be negative when two of UAVs are getting further. Based on this inference, they check the chance for the occurrence of conflicts only when $\tau < 0$. In order to determine are the UAVs in a conflict condition, they set a *safety* threshold distance r_{safe}. When the magnitude of $\vec{r}_m$ is smaller than the threshold distance r_{safe}, there is a predicted conflict between two UAVs. As long as there is a conflict, a conflict resolution maneuvering should be accomplished. We can intuitively know which direction each UAV should choose to avoid conflicts in Fig. 2.2 and solve this problem.

2.2 Stochastic Approach

In this approach, the collision avoidance problem is formulated as the optimal control of a stochastic system. The *Markov Decision Process* (**MDP**) is a representative such approach by Temizer, Kochenderfer, and Kaelbling [39].

2.2.1 Markov Decision Process

MDP is named after Russian mathematician Andrey Markov who is best known for his effort on theory of stochastic process. In 1950s, the concept of *MDP* began to be discussed [7]. After Howard [18] published his book *Dynamic Programming and Markov Processes* in 1960, numerous research about MDP was spawned [33] [28] [19].

MDP is an extension of Markov Chain, it introduce additional concepts such as actions (allowing choice) and rewards (giving motivation). *MDP* is a discrete stochastic dynamic programming process where the state of the system changes randomly according to the current state and action. An *MDP* is defined as tuple $(S, A, P^a_{ss'}, R^a_{ss'}, \gamma)$, where S is the state space, A is the action space. By taking action a from state s, there is a probability of $P^a_{ss'}$ to transit to state s' and receiving reward , $R^a_{ss'}$. Finally, $\gamma \in [0, 1]$ is the discount factor used to prioritize early rewards against future rewards and the value is typically close to 1.

MDP seeks the automatic generation of collision avoidance algorithms given models of aircraft dynamics, sensor performance, and intruder behavior. By formulating the problem of collision avoidance as a Markov Decision Process for sensors that provide precise localization of the intruder aircraft, or a Partially Observable Markov Decision Process (POMDP) for sensors that have positional uncertainty or limited field-of-view constraints, generic MDP/POMDP solvers can be used to generate avoidance strategies that optimize a cost function that balances flight-plan deviation with collision. However, the computational time can be heavy. Moving to full three-dimensional motion in a discretized formulation will take the size of the state space beyond the range of existing solvers. Other representations for the state space and new types of solvers are currently investigated [39].

2.3 Linear Programming Approach

Linear programming is a mathematical method for the optimization of a linear objective function. In 1939, Leonid Kantorovich [20] developed the first linear programming problem to plan expenditures and returns during World War II. In 1990s, Bemporad [9] and Paul [42] proposed the same idea that the linear programming problem can be expressed as linear constraints upon a mixture of continuous and integer variables.

2.3.1 Mixed Integer Linear Programming

Here, the collision avoidance problem is formulated as a linear programming problem. The *Mixed Integer Linear Programming* (**MILP**) is one representative technique to solve the problem of UAV collision avoidance using linear programming. *MILP* has proven to be effective [34][37]. *MILP* provides a best outcome for a given mathematical model containing a set of linear constraints. To begin, a number of constraints must be specified to model the UAV's motion dynamics; additionally, constraints related to collision avoidance such as minimum separation distance must be provided. These constraints are then fed to commercial *MILP* solvers such as *APML* and *MATLAB*, which develop a best path for the UAV. Each aircraft's path is optimized based on the expected paths of the other competing but cooperative UAVs; therefore, an uncooperative UAV can wreak havoc on the carefully planned system.

MILP is computationally expensive. If the optimal solution takes too much time, *MILP* can solve the problem and provide *less* optimal solutions over a prescribed allocated time. As the allocated time increases, *MILP* provides better and better optimal solutions. In order to decrease the computational cost, a technique derived from *model predictive control* known as *receding horizon* is used [25]. Using a receding horizon approach, the UAV path is broken down into small chunks where the MILP model is solved for N time steps, which serve as input for the next N time steps and so on until the process is finished. While this approach minimizes the computational burden, since the program is solved in smaller blocks, obstacles outside one time block distance from the UAV will have no effect on the path of the UAV; therefore, when paths for the next time block need to be computed, the UAVs may be too close to avoid collision. Thus, when using the MILP approach, the goal is to minimize computation time while increasing the size of the time blocks used in computation.

2.4 Potential Fields Approach

This approach was proposed for the first time in 1986 by Khatib [22]. This method assigns magnetic or electrical charges of the same sign to UAVs and the opposite charges to destinations: based on physics laws, particles with the same charges will repel each other while being attracted by the destinations (opposite charge). [14] [26]

2.4.1 Total Field

Sigurd and How[38] investigated the feasibility to apply potential fields such that UAVs do not necessarily need to know the positions of all other aircrafts. The repulsive force between magnetic field generated by each UAVs allow them to avoid each other spontaneously. An additional magnetic sensor and local magnetic field generator are required on board in this approach. Therefore, each UAV can be treated as a magnetic dipole. The magnetic flux density $\frac{B}{|B|}$ in the xy-plane around a magnetic dipole is shown in Fig. 2.3. By generating local field, sensing total field, and subtracting own field, they are able to find a set of angles where the UAV's own field is orthogonal to the x-axis or y-axis in the UAVs reference frame; these angles are shown in Fig. 2.4

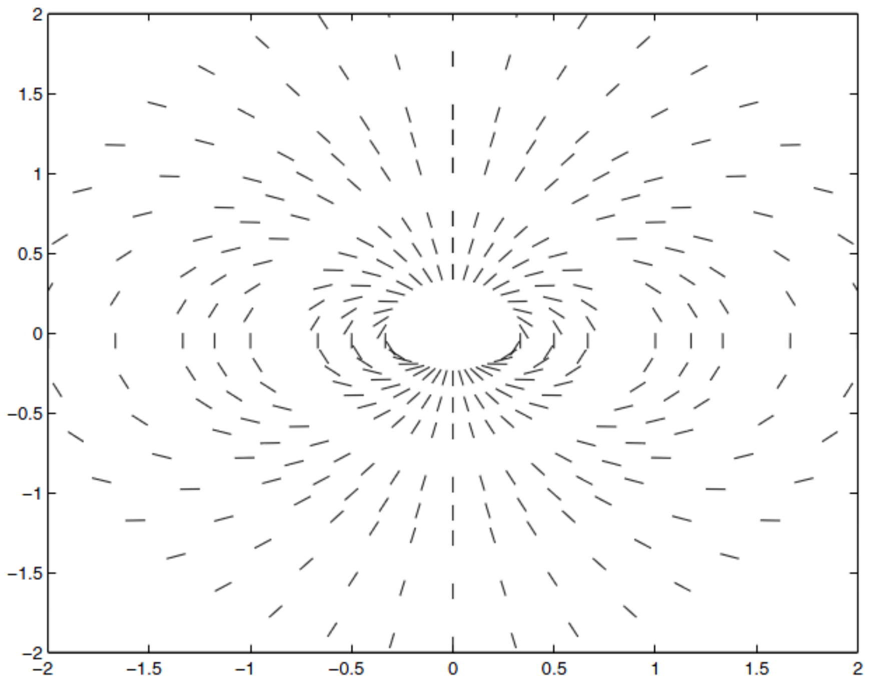
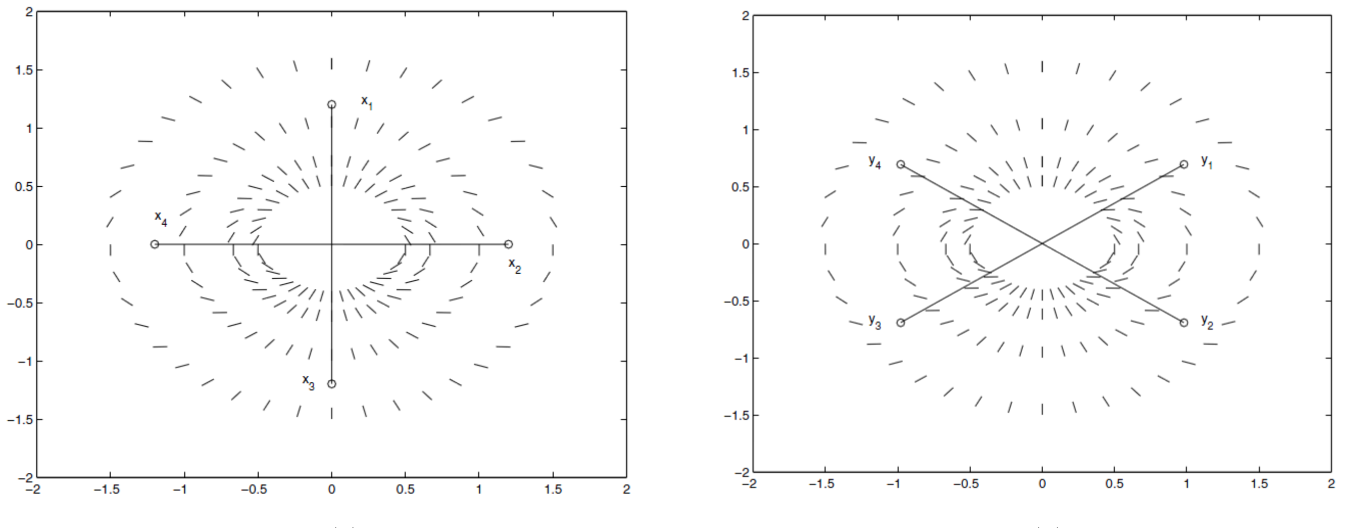


Fig. 2.3: Magnetic flux density around a magnetic dipole [38]



(a) (b)

Fig. 2.4: (a) Surrounding field along the x-axis (b) Surrounding field along the y-axis [38]

2.4.2 Force Field

Liu, Wang, and Dissanayakes [41] present research on artificial force fields in robot collision avoidance. They solved the path finding problem for an omnidirectional class of vehicles. By modeling goal waypoints as positive charges and other robots in the project space as negative charges, optimal routes were found by calculating a total force from the attractive force of the robots towards their destination and repulsive force away from other vehicles operating in the area. One particularly important innovation is the use of the elliptical potential fields, which results in less effort for the robot to continue forward since the artificial field generated by a robot extends further into the space ahead of it. Another important innovation includes the use of the expanding force fields; as a robot moves closer to its goal waypoint, the artificial field increases in size to ward off other robots in the area.

2.5 Grid-based Approach

As the name suggests, grid-based approach discretized the air by dividing it into discrete square or cubic cells of uniform size. The collection of cells are represented by a weighted graph in which *vertices* are the cells and the *edges* connect adjacent cells. On a plane, a vertex has eight edges: orthogonal (north, west, south, and east) and diagonal (northwest, southeast, south east, and northeast). Weights of high (or infinite value) on the edges can be used indicate an obstacle. Such a graph representation allows the use of short path algorithms such as Dijkstra's [10] or Bell-Ford [8] [13].

2.5.1 Search Trees Problem

In 1959, Dijkstra discussed the minimum cost problems in connection with graphs. Considering n nodes, and there are branches connecting them with given lengths. Assuming one node has at least one path to any other nodes,the path with minimum length between given two nodes P and Q can be found by the following step:

1) Create three sets for the branches:I, II, III, and three sets for the nodes:A, B, C.

2) Place all the nodes in set C, and all branches in set III.

3) Move the starting node P to the set A, and set this node as current node.

4) Consider all branches r connecting the current node with nodes$R$. If the i_{th} node R_i belongs to set B, we consider does the use of branch r_i provides a shorter distance than the use of corresponding branch in set II. If it does, branch r_i replaces the branch in set II. Otherwise, the branch r_i is rejected and transfered to set III.

5) If the i_{th} node R_i does not belong to set B, we transfer the nodes R_i to the set B and the branch r_i to the set II.

6) If we follow the rule that we must use branches from set I and one branch from set II, then every node in set B can connect to node P in only one way. Therefore, we can compute the distance between node P with each node in set B. We transfer the node with minimum distance from P to set A and the corresponding branch to set I. Make this node as current node.

7) Return to step 4 and repeat the process until node Q is transfered to set I, then the nodes in set A is the solution.

Dijkstra's algorithm is capable to solve problems which can be reduced to the minimum path finding in a weighted graph. The graph does not necessarily need to assign the same weight (length of a branch) for the branches. Moreover, the weight can vary with different direction in which it is traversed. One restriction for Dijkstra's algorithm is that it does not work on a graph with negative weights. An algorithm with similar structure named Bell-Ford algorithm [8] [13] is considered when there are negative weights in graphs.

Although Dijkstra's algorithm has great capability to solve the minimum cost problem, given the large number of nodes, the computation can get prohibitive. A few years later, a generalization of Dijkstra's algorithm was proposed to reduce the nodes that must be explored.

2.5.2 Heuristics

In 1966, the research for the actual construction of a mobile robot nicknamed "Shakey" [1] was conducted by SRI International (then Stanford Research Institute). Fig. 2.5 is a picture of Shakey, there is a television camera onboard so that Shakey is capable to capture images around it to detect those obstacles. The method for the navigation is to plan a sequence of way points so that Shakey can follow these way points from one place to another.

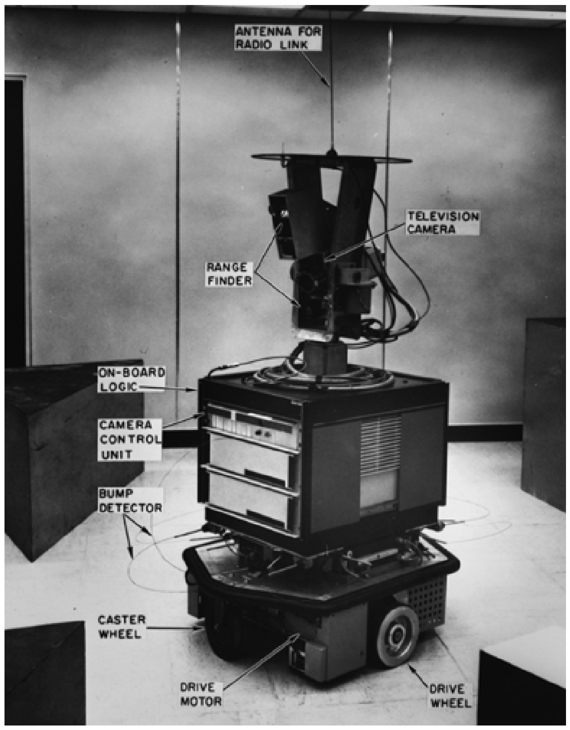


Fig. 2.5: Shakey, the mobile robot system. 1966 through 1972 [1](Photograph courtesy of SRI International.)

Shakey stored the positions of obstacles and itself in a "grid model" as shown in Fig. 2.6. They used smaller cells near the obstacles in order to describe positions more accurately.

Nilsson, a member of Shakey's research group, stated "This could be one of the first application of adaptive cell decomposition in robot motion planning and is now a commonly used technique." [31]. Shakey's navigation problems can be reduced to a search problem in a weighted graph, and numerous approaches for searching graphs were proposed during that period including Dijkstra's algorithm and Bell-Ford algorithm we mentioned before. These approaches were capable of solving Shakey's navigation problem and guaranteed the shortest path. However, they could be inefficient for the computation with complicated problem such as large number of nodes.

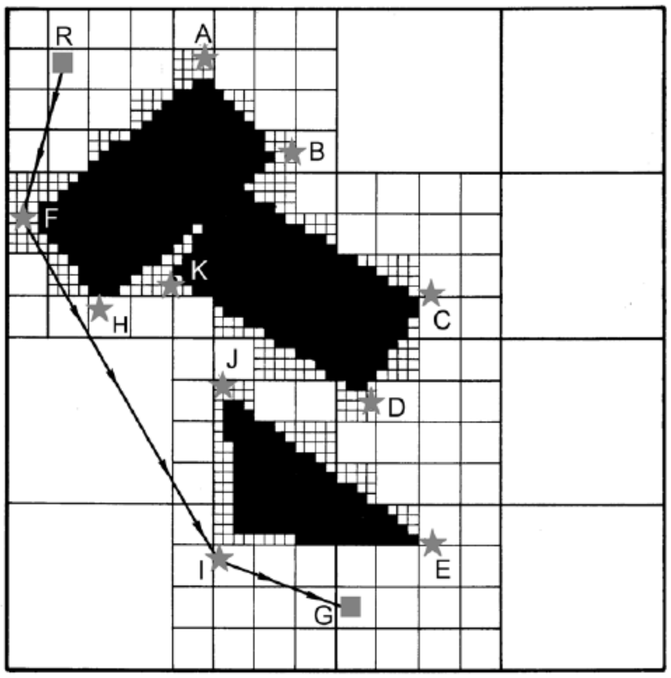


Fig. 2.6: Illustration of Shakey's navigation problem[31](Original image from SRI International)

Inspired by Doran and Michie's Experiments with the Graph Traverser program [11], where they define heuristic functions to assign costs based on the estimated difficulty to reach the destination to deal with the eight-piece, sliding-tile puzzle problem, Nilsson [31] reasoned that for a mobile robot navigation problem, a good heuristic function to estimate difficulty to reach the destination (before actually searching nodes) could be the *airline distance*, which means the length of path ignoring any intervening obstacles to reach the destination. Nilsson suggested to use this heuristic function as the weights of corresponding nodes in the search tree.

Raphael, the director of Shakey at that time, provide an improvement for Nilsson's idea for Shakey's navigation problem. Instead of only using the heuristic function as the weights of corresponding nodes in the search tree, Raphael suggested to combine the heuristic function Nilsson proposed with the distance robot already traveled so far from the initial position as the total weight of nodes. [36]

Hart, another member of Nilsson and Raphael's group at SRI, considered Nilsson and Raphael's algorithm, and indicated that if the remaining distance the heuristic function estimated was never larger than the actual remaining distance, then the path their algorithm find would be the optimal solution in this problem. Moreover, Hart believed their algorithm would be the most efficient procedures in the search tree problem comparing with other existing approaches that also guaranteed to find the optimal solution.

2.5.3 A^* Algorithm

In 1968, Hart, Nilsson, and Raphael [16] proposed a proven optimization of Dijkstra's algorithm dubbed A^* (after multiple earlier version A^1, A^2, etc)). A^* is an *informed search* algorithm: it first searches the routes that appear to most likely lead to the destination. For this, A^* uses a distance-plus-cost *heuristic* to determine the order in which to visit potential nodes on the route.A^* greatly improves the time of Dijkstra's algorithm and works well for path planning for robots navigating around fixed obstacles.

For collision avoidance for UAVs, other UAVs are the moving obstacles. Tooren, Heni, Knoll, and Beck[40] adapted and implemented A^* for UAVs as a search over nodes of motion primitives that are short trajectory segments. Each of these motion primitives is chosen such that it generates a flyable, smooth trajectory valid for the current state and the performance limits of the aircraft. Fig. 2.7 illustrates some expanded nodes resulting from typical motion primitives with their [G / H] cost noted in the attached boxes where G is the cost from the starting point to the current point and H is the *perceived* (heuristic) cost from the current point to the destination. Note that for simplicity, not all possible segments are shown, and the least cost path is drawn as a solid line. The set of possible elements includes typical flight elements such as linear segments, curve segments but also more complicated elements like (3D) Dubins sets [12]. The computational time varies greatly as the possible airspace changes. As the possible airspace grows, it is clear the computational complexity increases greatly. On the other hand, if the airspace is broken into fewer, larger cells, there might not be enough detail to plan efficient non-colliding paths.

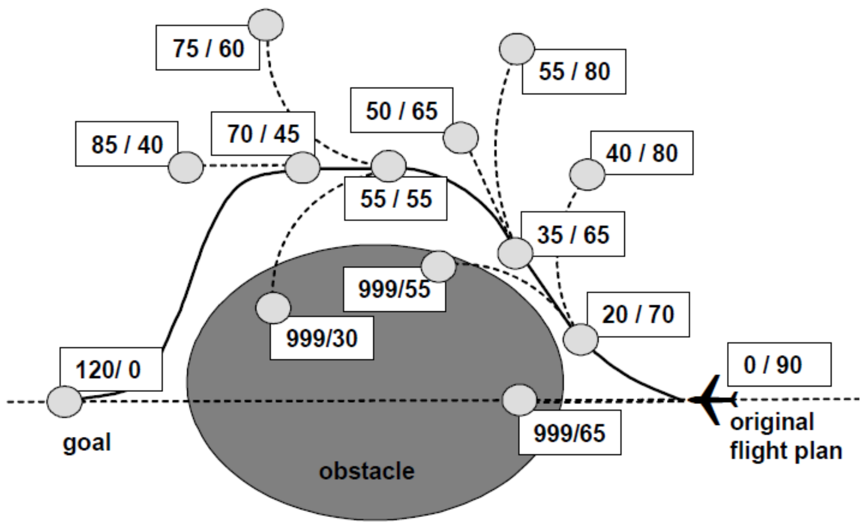


Fig. 2.7: A^* path search around an obstacle.[40]

2.6 Conclusion

In a dynamic environment, the computational time is critical. The computational time of A^* varies greatly as the possible airspace varies (overall field size and cell size). The complexity of A^* can be reduced by designing a good heuristic function so that A^* has more flexibility to adapt different scenarios. The advantage of A^* is that it can be flexible by allowing different heuristic functions to handle different situations. A^* is a good candidate to perform real time path planning.

Chapter 3

UAVs Collision Avoidance

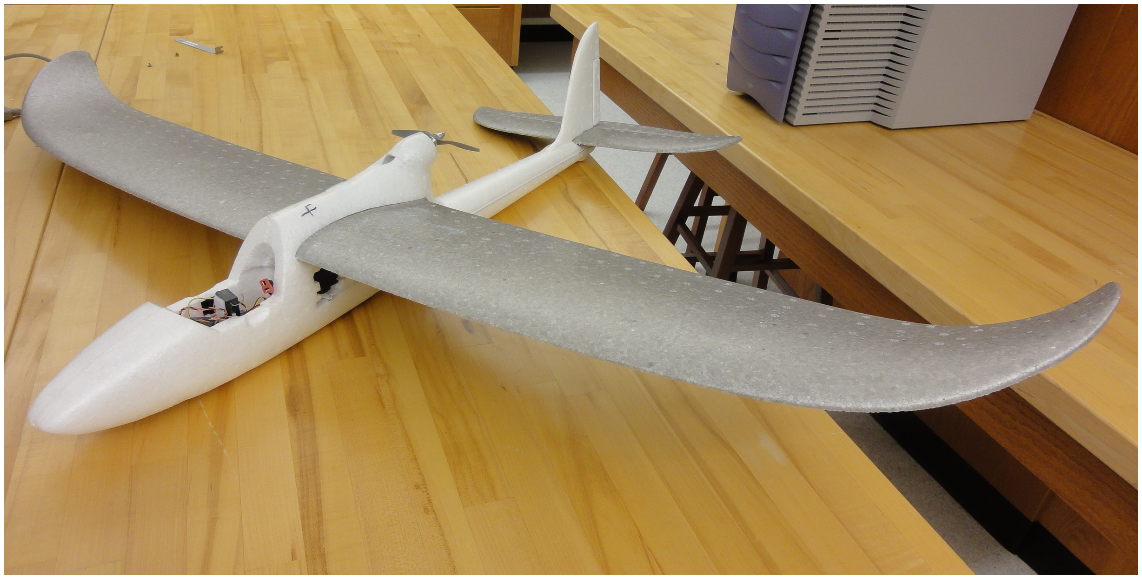


Fig. 3.1: Easy Star, the Unmanned Aerial Vehicle

3.1 Easy Star

The approach we will propose in this chapter will be implemented on the UAV platform named *Easy Star* (see Fig. 3.1). Easy Star is a fixed-wing UAV, consisting of the body, a wing, a rudder, an elevator, a motor, and carving layout hosting the electronics such as *Arduino* [2] on board. Where as the motor controls the speed of Easy Star, rubber, elevator, and wing control the flying direction. *Arduino* on board is capable of transmitting and receiving information with the ground station (such as a laptop) or other UAVs.

3.2 Conflict Detection

A conflict can be defined as a predicted violation of a separation assurance standard. Under this definition, a conflict exists as long as two aircraft will encounter within the safety distance at some time in the future [21]. In our approach, we predict the positions of UAVs in the coming future and use these information to avoid conflicts by applying them in the heuristic function of A^* algorithm. Then, we can achieve *intruder awareness*.

3.2.1 Intruder Awareness

We call *intruder awareness* the estimated danger we consider in the heuristic function. Fig. 3.2 illustrates for how to achieve this objective of intruder awareness.
First we consider the difference of the x coordinates of the UAV and the intruder:

$$x_{01} = x_1 - x_0 \tag{3.1}$$

Where the sign of x_{12} tells whether the intruder is on the right or the left of the UAV. In Fig. 3.2, we can see the sign of x_{01} is positive, so the intruder is on the right of the UAV.

Second , the velocity of UAVs can be calculated by differentiating the movements. As long as the velocity is known, the UAV bearing is also known. Then, we can tell whether the intruder is approaching or leaving along the x-axis by checking the sign of x-axis velocity. Finally, we multiple the relative x-axis position and velocity: if the product is negative, then the intruder is approaching along the x-axis direction. In other words, the UAV is “in front of” the intruder. Applying the same process on the y direction, we can tell from which y direction the intruder is approaching.

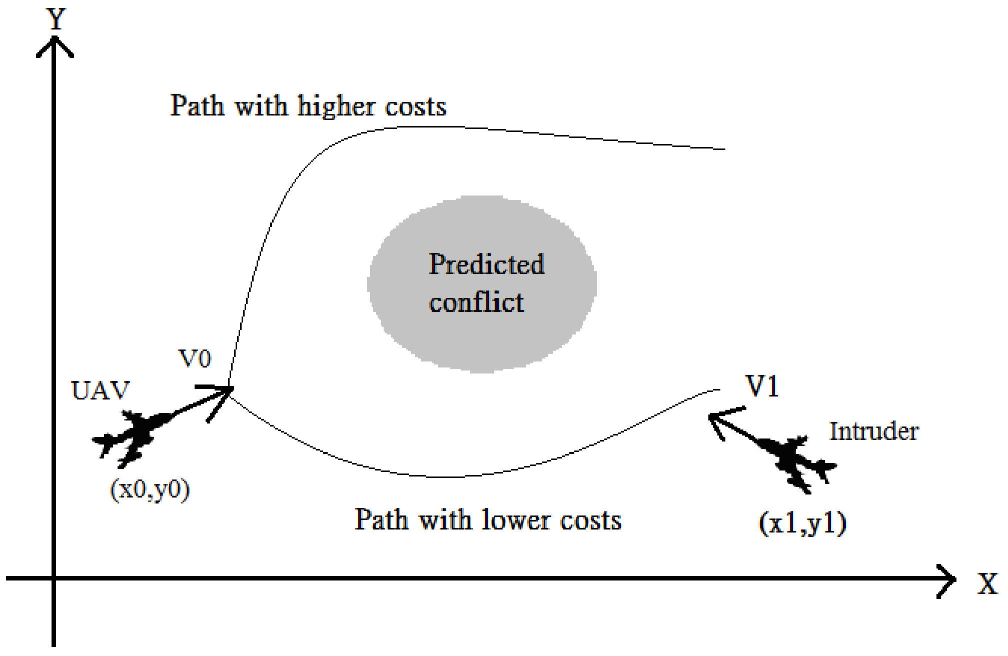


Fig. 3.2: Illustration of intruder awareness.

Third , given the predicted positions of the UAV and n number of intruders, we are able to estimate the danger in the future with the equation:

$$\sum_{i=1}^{n} |(X_i - X_0)| + |(Y_i - Y_0)| \tag{3.2}$$

The reason of using absolute value of x, y difference instead of the actual distance is that it makes more sense to the grid-based approach like A^* algorithm. Because every unit of the x,y difference can be treated as moving one grid. The *intruder awareness* will be applied in the heuristic function which is discussed in section **3.3.3**.

3.2.2 Conflict Determination

In order to detect conflicts and advise the pilot to avoid potential collision, a Traffic Alert and Collision Avoidance System **(TCAS)** [3] is commonly used in air traffic management administration including **FAA**. A **TCAS** defined different level of potential collision zones that should be protected around aircraft as shown in Fig. 3.3. **FAA** required all commercial turbine-powered transport aircraft with more than 30 passenger seats (or maximum take off weight above 33,000 lb/15,000 kg) to equip **TCAS II** since 1993. In our case for Easy Star, we refer to the definition of **TCAS** and the distance that UAVs can reach within 35 seconds as a threshold distance to activate the collision avoidance function. In our simulation, we assume the UAVs fly 5 meters per second so that the safety distance is 175 meters.

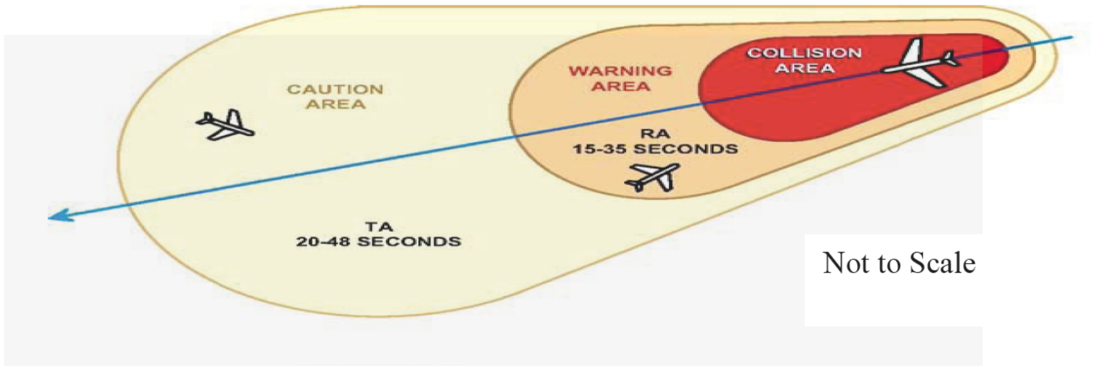


Fig. 3.3: Classification of protected zones, including Traffic Advisory Zone, Resolution Advisory Zone, and Autonomous Avoidance Zone.

3.3 Collision Avoidance

Once the potential conflict is detected, we activate A^* algorithm to plan a safe path to avoid intruders. Fig. 3.4 illustrates the architecture of the collision avoidance model. This model includes the *ArduPilot*, the collision detection function, and the A^* algorithm. The *ArduPilot* is an open source software that takes care of flying the UAV: given a waypoint, the ArduPilot flies the AUV to it. Our team designed the software to upload in flight new waypoints to the *ArduPilot*.

The *collision detection function* continually monitors all UAVs to detect an imminent conflict. Given the positions and the bearing of all UAVs, this function derives the velocities

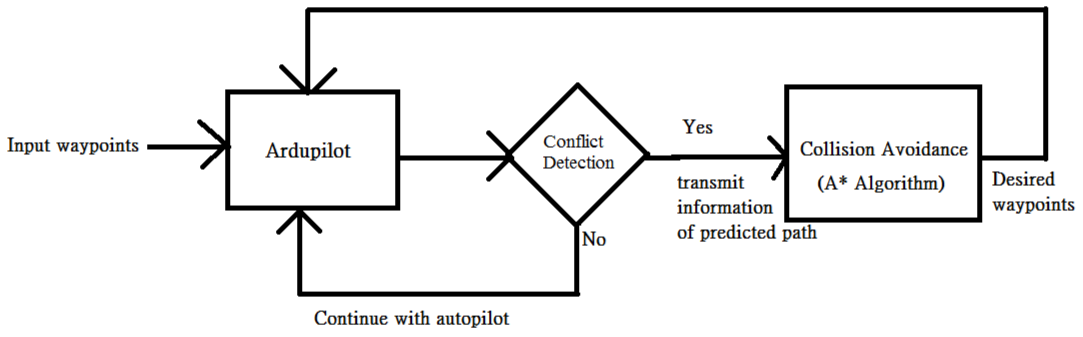


Fig. 3.4: Collision avoidance architecture

of all UAVs and can predict an imminent conflict or a collision. If a conflict/collision is predicted, the *collision detection function* activates A^* and provides to it the positions of the UAVs who may collide. The future positions of all UAVs are continuously computed 35 seconds ahead. The 35 seconds correspond to the responding time proposed by **TCAS**. However, the responding time can be modified based on the speed of UAVs.

When A^* starts, the *collision detection function* is stopped for 35 seconds. After 35 seconds, the collision detection function is reactivated again to update the positions and bearings to eliminate the discrepancies between $A*$ algorithm and the *ArduPilot*. As long as UAVs are too close (in the distance that UAVs can reach within 35 seconds), the A^* algorithm remains active. Concurrently, the predicted positions of intruders(*intruder awareness*) are fed to the A^* algorithm as the dynamic obstacles. Based on the most recent update, the $A*$ algorithm computes the new path to avoid collisions.

3.3.1 A^* algorithm

One feature of the A^* algorithm is the high flexibility of cost function. The equation of total cost is

$$F = G + H \tag{3.3}$$

where G is to the cost from the starting point to current (visited) one, and H is the heuristic function estimating the cost from the current point to the destination. Therefore, the design of G and H will determine the total cost F which can greatly affect the path A^* finds.

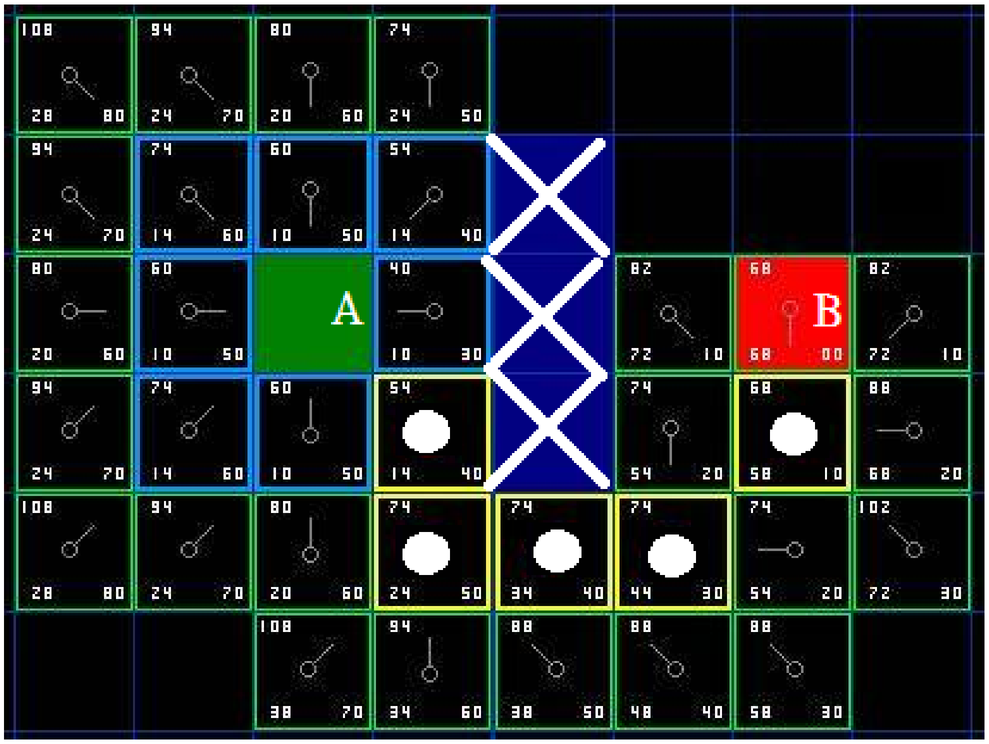


Fig. 3.5: Illustration of $A*$ path finding algorithm.[27]

Once there is an obstacle that makes the current path cost more than other available path, A^* will go back to the new node of lowest cost and expands its neighbors. After repeating this finding process, A^* will finally find a path that cost least once it expands to the destination. As shown in Fig. 3.5, the A^* algorithm starts expanding nodes from A to the goal B, and the grids with cross marks are the obstacles.

Each expanded grid contains the information: F at the top left, G at the bottom left, H at the bottom right, and the arrow pointing its parent. In this example, the heuristic function is the *Manhattan* method:

The *Manhattan* method [6] calculates the total vertical and horizontal distance between the current cell and the destination, ignoring obstacles and discarding any diagonal movement. In fact, this is what we used to estimate the danger in *intruder awareness* (section **3.2.1**) .

3.3.2 Dangerous Grids

In order to deal with dynamic environment, we extended A^* algorithm with one approach we called **dangerous grids** as shown in Fig. 3.6.

The concept of dangerous grids is to predict the positions of UAVs which we already did in the *intruder awareness*, and reconstruct the map in a discrete time system. In Fig. 3.6, the UAV and intruder is approaching each other by the increasing time step. Those colored grids indicate that the additional costs of heuristic function will be assigned in that time step when the colored grid is expanded by A^*. Therefore, A^* is capable to construct the map with dynamic obstacles and avoid those grids with higher cost.

3.3.3 Heuristics

The heuristic function H is critical to determine the performance of the A^* algorithm. By using different heuristic functions, the result can be completely different. Fig. 3.7 shows how the paths differ by using two different heuristic functions.

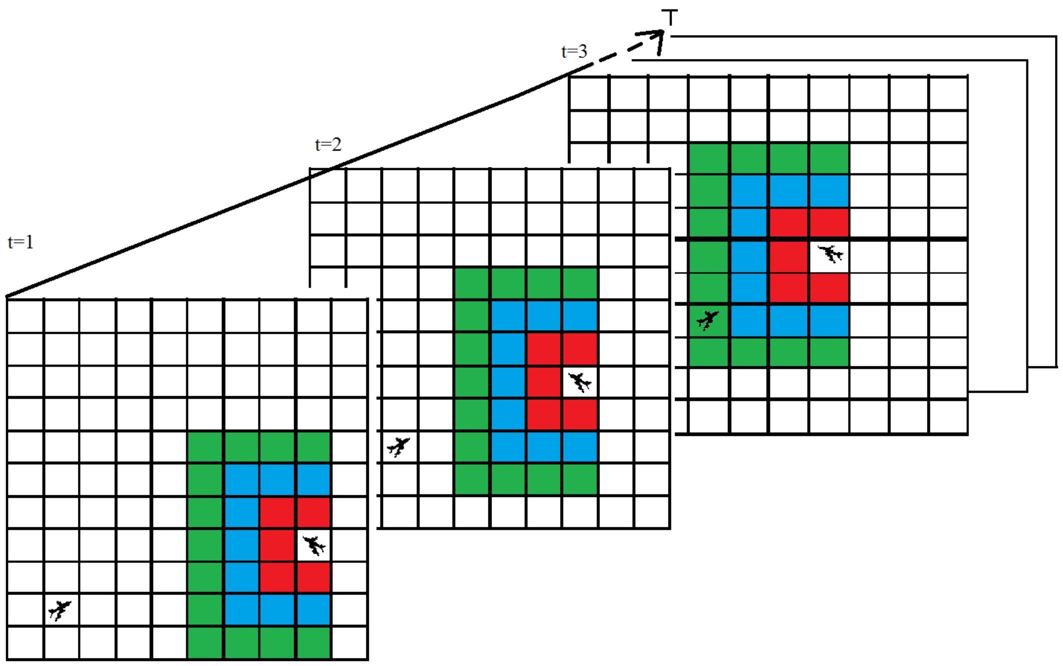


Fig. 3.6: The dangerous grids changes with the different time step.

In order to implement A^* algorithm in a UAVs path planning problem, we made some modification. **First**, normally there should not be any obstacle in the sky so that every grid is theoretically walkable in our problem. Instead, we have other aircrafts in the air as moving obstacles in our problem. **Second**, except the Manhattan method, the heuristic function has to be improved to assign the costs to those paths which are too close to intruders. Therefore, in this study the cost estimation includes the Manhattan distance and *intruder awareness*.

This study introduces two different types of heuristic functions. The first is conservative heuristics: it respects *rules of the air* [5] to ensure aircraft's safety. The second is aggressive heuristics: it takes risks to pass through intruders in order to arrive destination faster. Both conservative and aggressive heuristic functions are based on the combination of *Manhattan* method and *intruders awareness*.

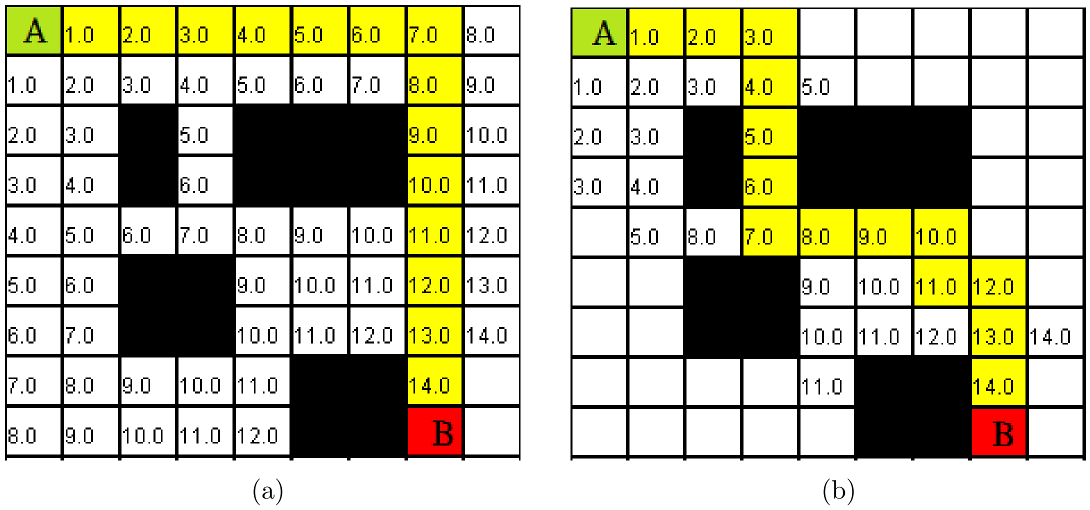


Fig. 3.7: (a) Path finding using classic $A*$ method (b) Path finding using fudge method [29]

In the conservative heuristic function, we assign different costs to follow some safety rules. For example when a conflict(the distance that UAVs can reach within 6 seconds) is detected, the path approaching the intruders should cost more than the path leaving the intruders. Also, in order to reduce the potential conflict after the avoidance, we assign lower cost for the path passing behind the intruders to encourage the UAVs to choose a safer path. Recall that in the *intruder awareness* section, we used a mathematical method to judge a intruder is approaching or leaving and which direction does it come from. Now, we can apply this method to conservative heuristic function:

$$H(t) = \sum_{i=1}^{n} \alpha_1(|X_0 - X_g| + |Y_0 - Y_g|) + \beta_1(\lambda - |(X_i - X_0)| + |(Y_i - Y_0)|) \tag{3.4}$$

Where $|X_0 - X_g| + |Y_0 - Y_g|$ is the *Manhattan* method, α, β and λ are constants. The current (visited) grid is X_0, Y_0, and the destination is X_g, Y_g. Finally, X_i, Y_i is the intruder. If the intruder is approaching, assign more penalty for those grids moving toward the intruder by increasing the value of α, β, in the heuristic function, then the cost will be larger. Otherwise, if the intruder is approaching but the current grid is moving away from it, assign smaller penalty. Other conditions are also considered by adjusting suitable α, β. No matter in what situation, λ remains the same so that when the intruder is approaching, where $|(X_i - X_0)| + |(Y_i - Y_0)|$ gets smaller, the total cost will get larger. The value of α should be alway larger than β because *Manhattan* method contributes the remaining length of the path, and we should weight it to assure A^* is searching toward to the destination. Otherwise, the path may be trapped when there are too many intruders in the way to destination.

In the aggressive heuristic function, we try to find the path is most efficient and maybe a little more dangerous. Therefore, we only consider the distance between UAVs to ensure the minimum requirement of safety. The aggressive heuristic function

$$H(t) = 2 * fieldsize - \sum_{i=1}^{n} |(X_i - X_0)| + |(Y_i - Y_0)| + (|X_0 - X_g| + |Y_0 - Y_g|) \qquad (3.5)$$

Where X_0 and Y_0 is the position of our UAV,X_i and Y_i are the positions of intruders. Similarly, when the distance between UAVs is getting closer, the summation part in the equation is getting smaller, and the cost function $H(t)$ is getting larger. The cost of heuristic function increases when the value *intruder awareness* decreases. Therefore, intruder awareness provides information about how far are the intruders and whether they are approaching or leaving.

Note that for the conservative heuristic function, we consider every intruder independently and compute the cost respecting to each intruder. Then, we sum up all the cost. In this way, we carefully consider each intruder's position and movement. For the aggressive heuristic function, we consider all intruders at one time so that as long as the path is not too close to the intruders, it is acceptable.

A^* is often implemented as a search over a grid space. However, this is not necessary for A^* and in the case of conflict avoidance for aircraft it is actually more convenient to implement it as a search over nodes of motion primitives, - namely, the short trajectory segments [35]. We should not consider the grid space as the only way to implement A^*. However, using a grid space is more convenient to program because a grid space describes more easily the map, especially in the two-dimensional case. For example, when we use a grid space to search over, we can easily treat one UAV as one grid so that as long as we make sure the path does not pass through that grid, we can guarantee conflict avoidance. Otherwise, when we use floating points to describe the UAVs, we need to work more accurately and consider the shape of UAVs to determine what points should be left to pass through.

Chapter 4

Simulation

In this chapter, we will investigate the performance of both conservative and aggressive heuristic functions of A* algorithm in the UAVs collision avoidance problem. We will show the simulation results and discuss the influences by using different heuristics.

4.1 The Simulation Setup

We consider the following metrics to evaluate A^*'s performance:

a) **Deviation ratio** $= \frac{\text{ActualDistance}}{\text{MinimalAchievableDistance}}$. The minimal achievable distance is the length of the shortest path that UAVs can travel when there is no obstacle in their way. In other words, the shortest path is the straight line between start point and the destination. The actual distance means the total length of the path that the A* algorithm proposes to avoid obstacles in the air.

b) **Number of conflicts**. This is the number of pairs of planes within 6 grids. The way we count the conflicts is to see how long have the UAVs been within conflict distance(6 grids). Therefore, if two UAVs keep flying within 6 grids, the number of conflicts will keep counting with time until the conflict is over.

c) **Number of crashes**. This is the number of pairs of planes within thrice the wingspan of a plane. In our simulation, we define there is a crash when two or more UAVs' paths are in the same grid at the same time step. Once a crash occurs, the UAVs involved in the crash are eliminated, and the simulation will continue with the remaining UAVs.

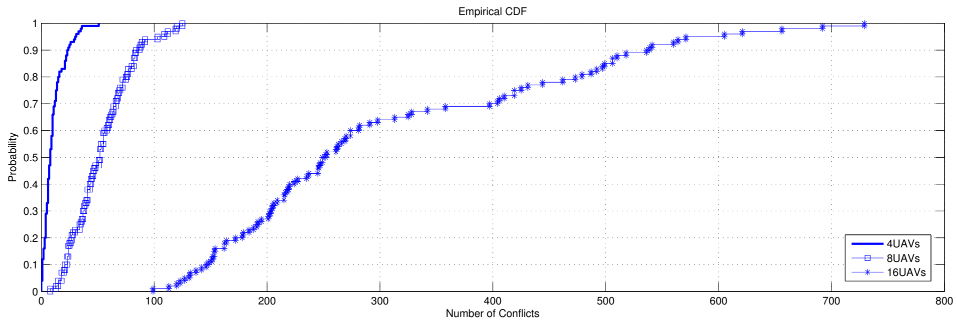
We assume that it takes the UAV one second to traverse a 5 $m \times 5$ m grid. By our definition of conflicts, a *conflict* occurs when two UAVs are within 6 seconds from each others. Therefore, We have at least 3 seconds to avoid a collision.

To evaluate A^*, we consider different configurations. A configuration consists of a fixed number of UAVs (2, 4, 8, or 16) and the size of the field (500 $m \times 500$ m and 1000 $m \times 1000$ m). We randomly create 100 different scenarios. Each scenario consists of four randomly picked waypoints per UAV. Each UAV must visit these four waypoints in a predetermined order. To generate random waypoints, we pick a random point in the field as the start point, then we use a certain radius 50 meters and 7 random directions to create the next waypoint. In this way, we can make sure that each UAV will travel at least 200 meters in the simulations.

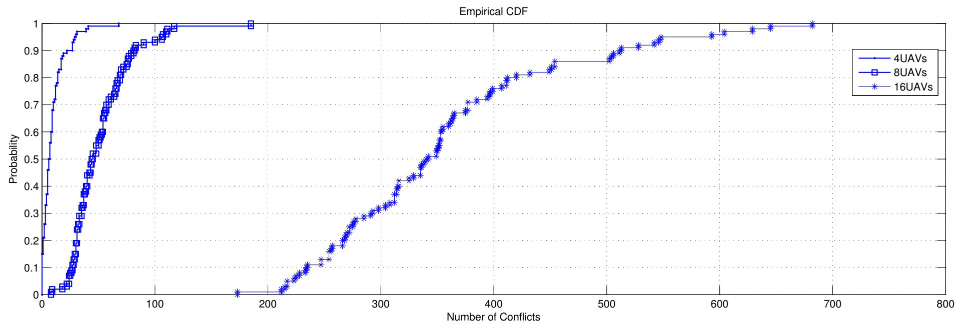
Given the same scenario, we simulate it on different configuration (number of UAVs, filed size) to evaluate $A*$ algorithm. We compare two different heuristic functions: *conservative* heuristic function and *aggressive* heuristic function. The conservative heuristic function assigns additional cost when the path of UAV cuts off intruders so that we expect the path to be circuitous to keep intruders in distance. The aggressive heuristic function only consider the length of path and the positions of intruders, so we expect it to try to find a feasible path around the intruders in order to reach the destination faster.

4.2 Simulation in a 500 $m \times 500$ m field

As shown in Fig. 4.1:(a), the *cumulative distribution function*(**CDF**) describes the probability that the number of conflicts is equal to or less than a certain value. For example, if we consider a straight line that the number of conflicts equals to 100, we can tell that the probability of 100 conflicts or less is 100% for 4 UAVs, about 96% for 8 UAVs, and only 1% for 16 UAVs. Comparing this with Fig. 4.1:(b), the CDF are similar for 4 UAVs and 8 UAVs. According to this observation, we consider that the abilities of avoiding conflicts are almost the same between these two heuristic functions when the number of UAVs is smaller than 8. In the scenarios of 16 UAVs, we observe that aggressive heuristics encounter more conflicts than conservative heuristics if we compare the CDF when number of conflicts equals to 200.



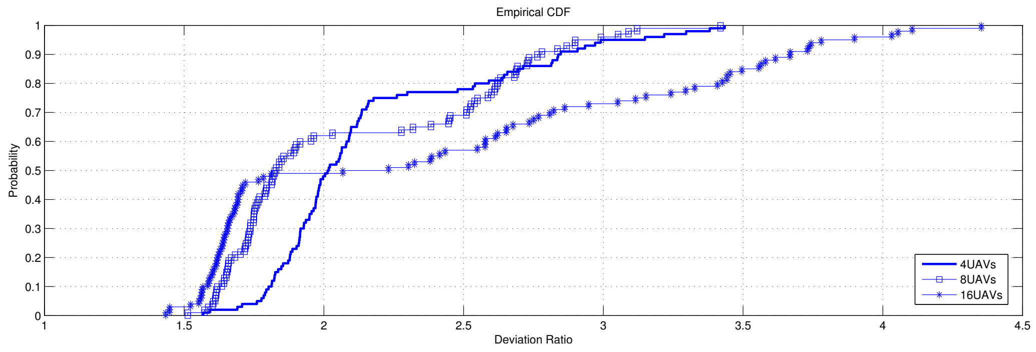
(a) conservative heuristics



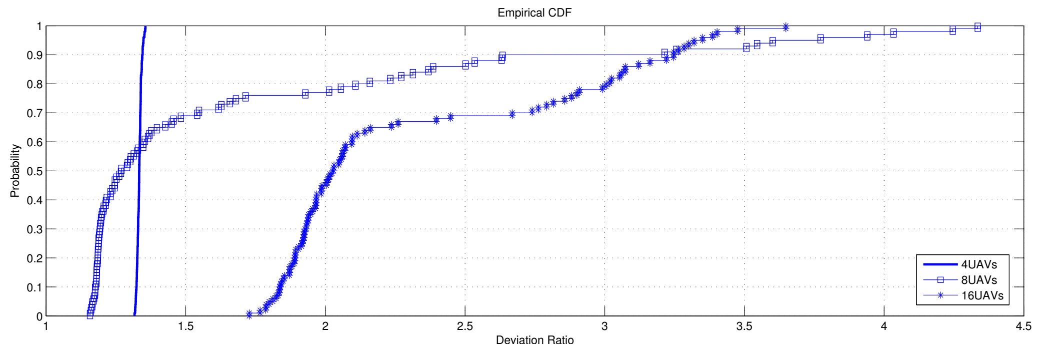
(b) aggressive heuristics

Fig. 4.1: Cumulative density function(CDF) plot of conflicts in a 500 $m \times 500$ m field

In our expectation, the deviation ratio of aggressive heuristic function should be smaller than the conservative one since it is supposed to find a shorter path. In Fig. 4.2, we can see that deviation ratios of aggressive heuristic function are approximately 1.5 times smaller than the conservative one when flying 4 UAVs or 8 UAVS. For the scenarios of 16 UAVs, when the probability reaches to 99%, the deviation ratio equals to 3.5 while using the aggressive heuristics and equals to 4.1 while using conservative heuristics. Comparing the difference between flying 8 UAVs with flying 16 UAVs, when we fly UAVs in a field with higher density ($\dfrac{16 UAVs}{500 \text{ m} \times 500 \text{ m field}}$), the advantage of aggressive heuristics with smaller



(a) conservative heuristics

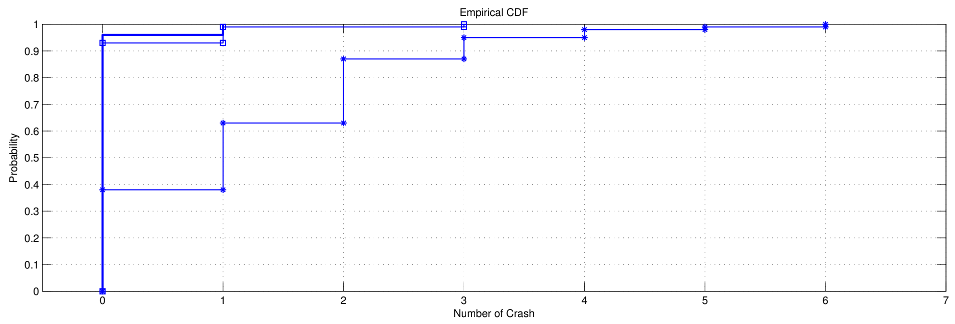


(b) aggressive heuristics

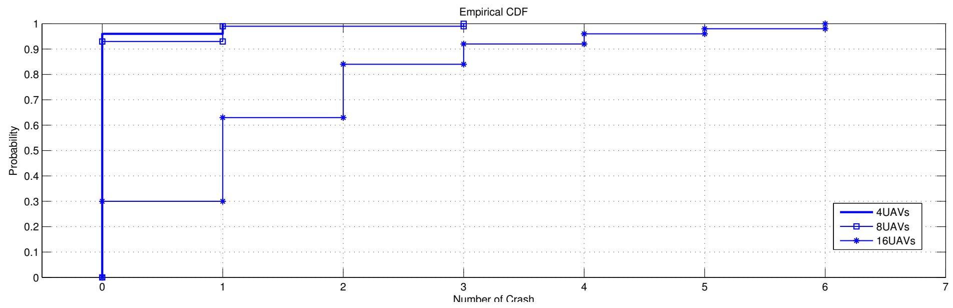
Fig. 4.2: CDF plot of deviation ratio in a 500 m × 500 m field

deviation ratio becomes unapparent.

Although the aggressive heuristics has higher probability to encounter conflicts, it usually have higher capability to find a shorter path. Because of the way we count conflicts is to see how long have UAVs been within conflict distance, it is possible for the aggressive heuristics to reduce the conflicts by shorten UAVs' flight duration in the air. Also, we observe in Fig. 4.2 that the average deviation ratio of aggressive heuristics is smaller in this configuration. However, in the scenarios of flying 8 UAVs, the maximum deviation ratio of aggressive heuristics is almost 1.5 times the ratio of conservative heuristics, this implies that in some extreme cases the path aggressive heuristic found is somehow much more inefficient than the path conservative heuristics found.We will make a further discussion for the reasons causing this unexpected phenomenon at the later part of this chapter.



(a) conservative heuristics



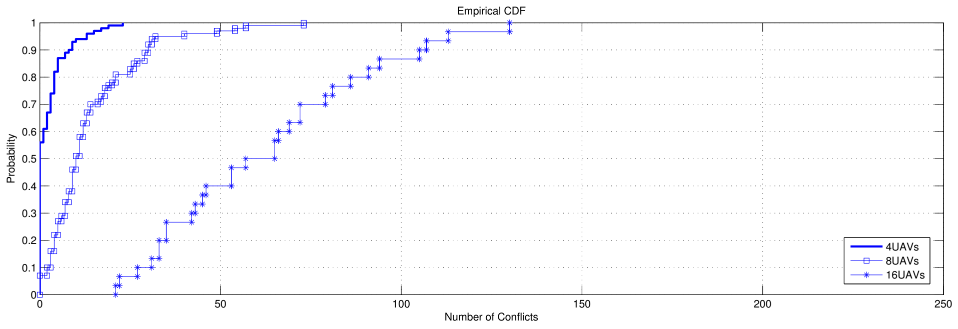
(b) aggressive heuristics

Fig. 4.3: CDF plot of crash in a 500 $m \times 500$ m field

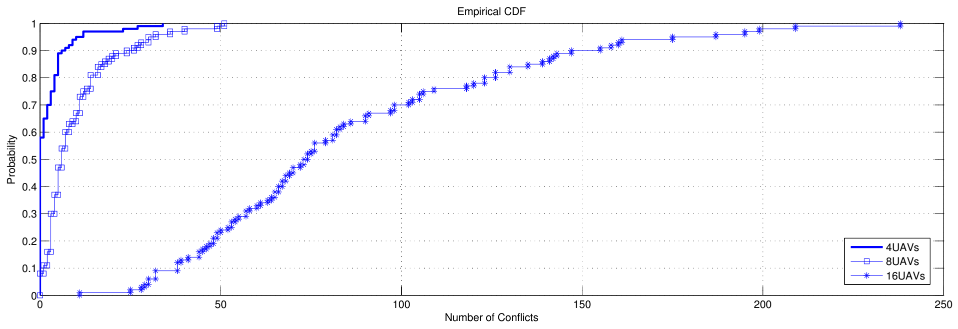
Fig. 4.3 shows that the ability of avoiding crash is similar to the ability of avoiding conflicts. When the number of UAVs is smaller than 8, the crashes are lower than 4 in 100 simulations. Actually, over half of the simulations result with no crash. However, when the number of UAVs is larger than 8, the crash increases gradually. In the worst case there are only 3 UAVs survived. The performances of conservative heuristics and aggressive heuristics are similar when the number of UAVs is smaller than 8. Once the number of UAVs is over 8, the aggressive heuristics tends to cause more crashes in this configuration.

4.3 Simulation in a $1000\ m \times 1000\ m$ field

In this section, we use a square field with 200×200 grids to simulate 2, 4, 8, or 16 UAVs fly at the same time. As a result, the performance of avoiding conflicts is better than the simulation in a $500\ m \times 500\ m$ field, especially when the number of UAVs is 16 as shown in Fig. 4.4. When flying with 16 UAVs, the conflicts are greatly reduced in this configuration no matter which heuristics we use because the bigger field provides UAVs more spaces to avoid conflicts.



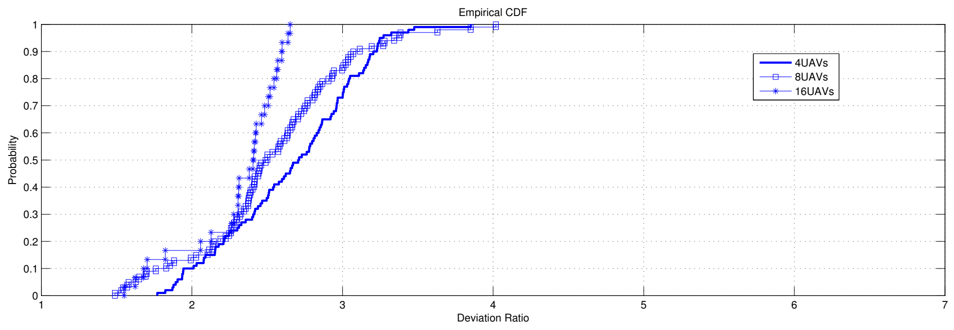
(a) conservative heuristics



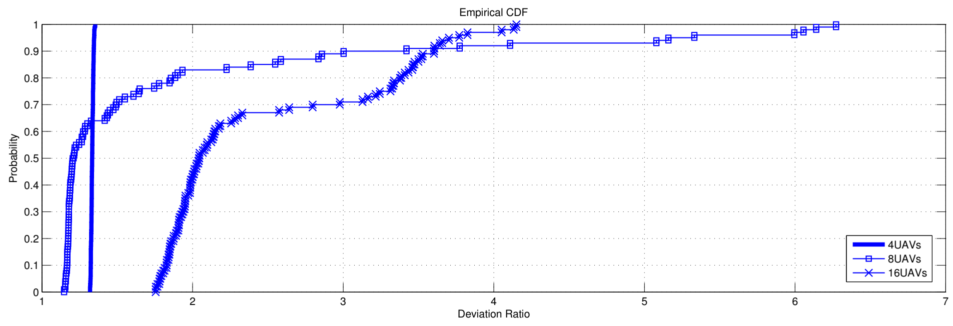
(b) aggressive heuristics

Fig. 4.4: CDF plot of conflicts in a $1000\ m \times 1000\ m$ field

The performance of deviation ratio is also better in the simulation in a $500\ m \times 500\ m$ field as shown in Fig. 4.5, except the scenario of 8 UAVs using aggressive heuristics. In our expectation, the aggressive heuristics should propose a path with lower deviation ratio, in this scenario we can see that even though the average deviation ratio is still smaller than conservative heuristics, in some cases the deviation ratio is as large as 6. Note that in the simulation of 500 $m \times 500$ m field, the deviation ratio of aggressive heuristics also has some unexpected behavior in the scenarios of 1 UAVs. This phenomenon raises one question: Why does this occur exclusively in the scenarios of 8 UAVs? To answer this, we need to review the relation between number of UAVs and the evaluation metrics. In the following section, we will discuss this issue with some figures to support our argument.



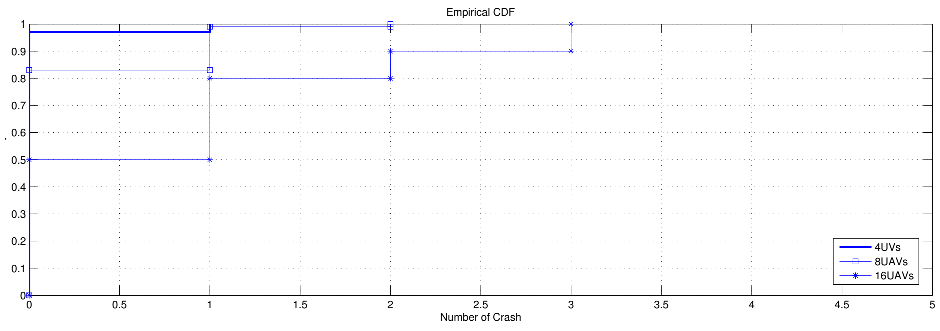
(a) conservative heuristics



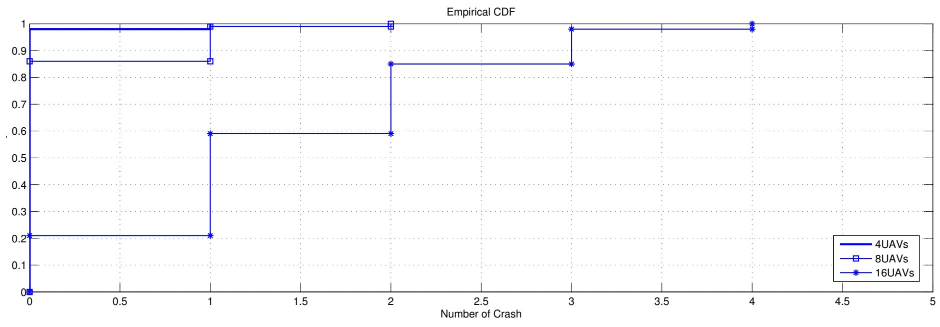
(b) aggressive heuristics

Fig. 4.5: CDF plot of deviation ratio in a 1000 $m \times 1000$ m field

The number of crashes in a 1000 $m \times 1000$ m field does not change a lot in the scenarios of 4 or 8 UAVs. However, in the case of 16 UAVs, both heuristics achieve a much smaller number of crashes as shown in Fig. 4.6 because the UAVs can avoid each other more easily in a bigger field. Also, it is not obvious but the crashes of aggressive heuristics are slightly more than for conservative heuristics as we expected.



(a) conservative heuristics



(b) aggressive heuristics

Fig. 4.6: CDF plot of crash in a 1000 $m \times 1000$ m field

4.4 Relation Between Number of UAVs and Metrics

In the scenario of 8 UAVs, we found that the aggressive heuristics may fail in some extreme cases. In order to confirm our thought, we review the relation between number of UAVs and metrics to compare these two heuristics as shown in Fig. 4.7 and Fig. 4.8. After the number of UAVs is larger than 8, the number of conflicts and crashes using aggressive heuristics increase considerably. No matter what size of the field is, we can observe that there are more conflicts and crashes by using aggressive heuristics. However, this still doesn't answer why the extreme cases only occur in the scenarios of 8 UAVs. We need to review the relation between number of UAVs and the deviation ratio.

The deviation ratio have interesting changes when the number of UAVs increase as shown in Fig. 4.9. The ratios of conservative heuristics does not have significant difference, and the ratios of aggressive heuristics have trends to decrease from 2 UAVs to 8 UAVs. The

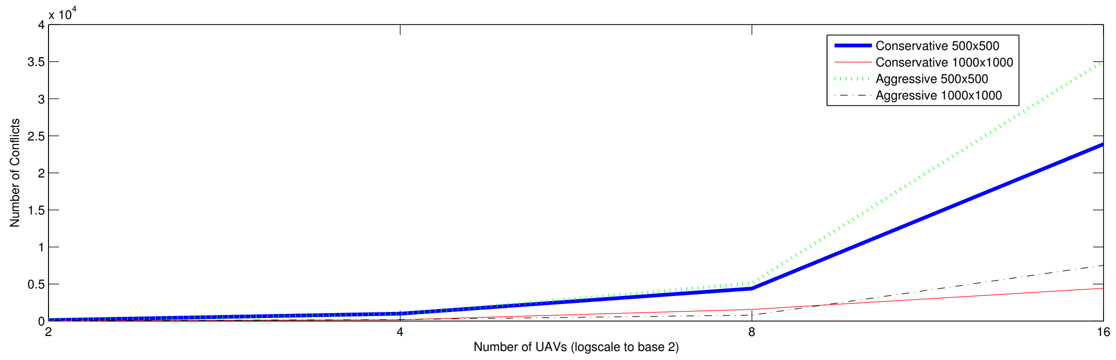


Fig. 4.7: Number of conflicts by number of UAVs

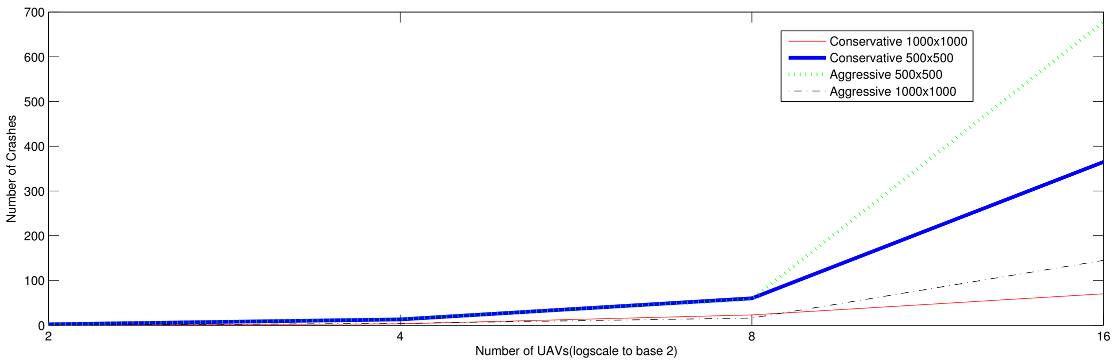


Fig. 4.8: Number of crashes by number of UAVs

reason may be the A^* over-reacts to the intruder because the aggressive heuristics assign higher costs to the path getting closer to the intruders. When there is only one intruder, only the direction toward to the intruder has higher costs so that the UAV may deviate too much. When there are more intruders from more than one direction, the deviation ratio becomes smaller because the A^* does not over-react to one certain direction.

Fig. 4.9 also presents the maximum and minimum deviation ratio in 100 simulations. We can see that the deviation ratio of aggressive heuristics does not change much for 2 or 4 UAVs. The scenarios with higher deviation ratio is usually due to the extreme cases. Therefore, the extreme cases with 2 or 4 UAVs may not have significant influences on the aggressive heuristics because they are more capable of finding feasible path when the field is

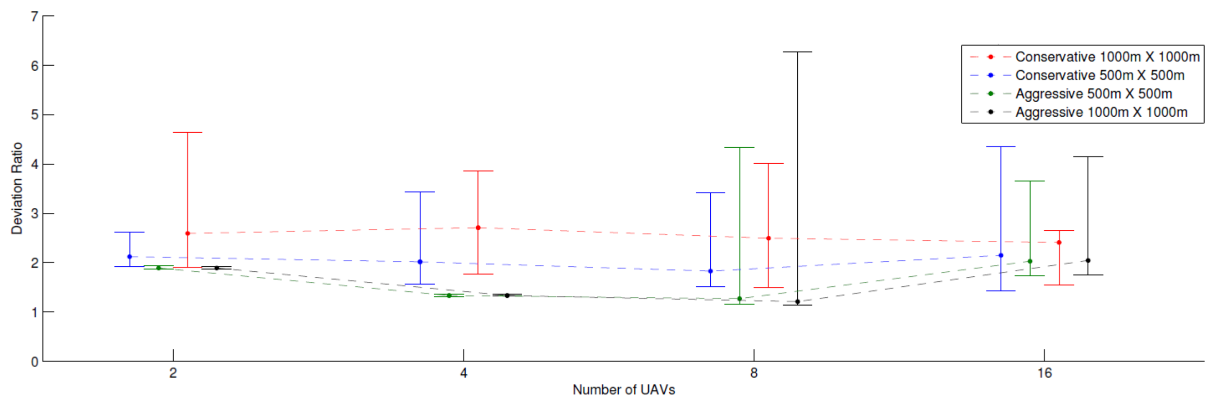


Fig. 4.9: Deviation ratio by number of UAVs

sparse.

When the number of UAVs is smaller than 8, aggressive heuristics always have lower deviation ratio. However, in the scenario of 16 UAVs, the ratio of aggressive heuristic becomes close to the ratio of conservative heuristics whether in a 500 $m \times 500$ m field or 1000 $m \times 1000$ m field. As a result, the aggressive heuristics lose its competitiveness in the scenarios of 16 UAVs. Both performances of number of conflicts and deviation ratio are worse than conservative heuristics. This raises another question: Why does aggressive heuristics fail in the scenarios of 16 UAVs?

Before answering this question, we have to discuss the previous issue about the extreme cases in the scenarios of 8 UAVs. From Fig. 4.9, 8 UAVs is a breaking point while the deviation ratio of aggressive heuristics increases. Thus the extreme cases in the scenarios of 8 UAVs may be treated as a portent of aggressive heuristics' poor performance in the scenarios of 16 UAVs. The reason of the exclusive occurrence with high deviation ratio in scenarios of 8 UAVs relates to the number of crashes. From Fig. 4.8, the crashes are much more in scenarios of 16 UAVs so that in these scenarios the UAVs don't even have a chance to avoid collision but crash. In the scenarios of 8 UAVs, it happens to have enough space for UAVs to avoid each other even with a huge deviation in some extreme cases. This explanation also implies the answer for the second question.

The aggressive heuristics fails in the scenarios of 16 UAVs because there is not enough space for collision avoidance. Note that the aggressive heuristics only consider the positions of intruders, it does not have the strategy like rules in the air [5] to avoid them. Therefore, when the air is crowed by too many UAVs, the aggressive heuristics loses its ability to plan a better path either by avoiding collision or by moving to the destination. It not only results in more conflicts and crashes, but also increases the deviation ratio.

4.5 Extreme cases for UAVs collision avoidance

After reviewing numerous simulations, we found some extreme cases that the UAVs may suffer in finding feasible path. By investigating these cases, we may be able to find some boundaries or limitations for the evaluation metrics of the UAVs.

One of the tough situations for collision avoidance is UAVs flying in parallel lines. In this case the path of UAV in the middle will be blocked by the other two UAVs from both sides. In general, the A^* algorithm can handle this situation by assigning different costs for left turn and right turn. However, in the example of Fig. 4.10, there are three UAVs parallel with each other at the beginning. Also, the initial position of the UAVs at the top are close to the boundary of the field so that there is no enough space for them to avoid each other. Moreover, the UAV from right also goes to the same line in order to avoid the UAV at the bottom. Even though the UAVs enter the same line at the top at different times, some of them fail to exit the line in time so that the crash occurs eventually.

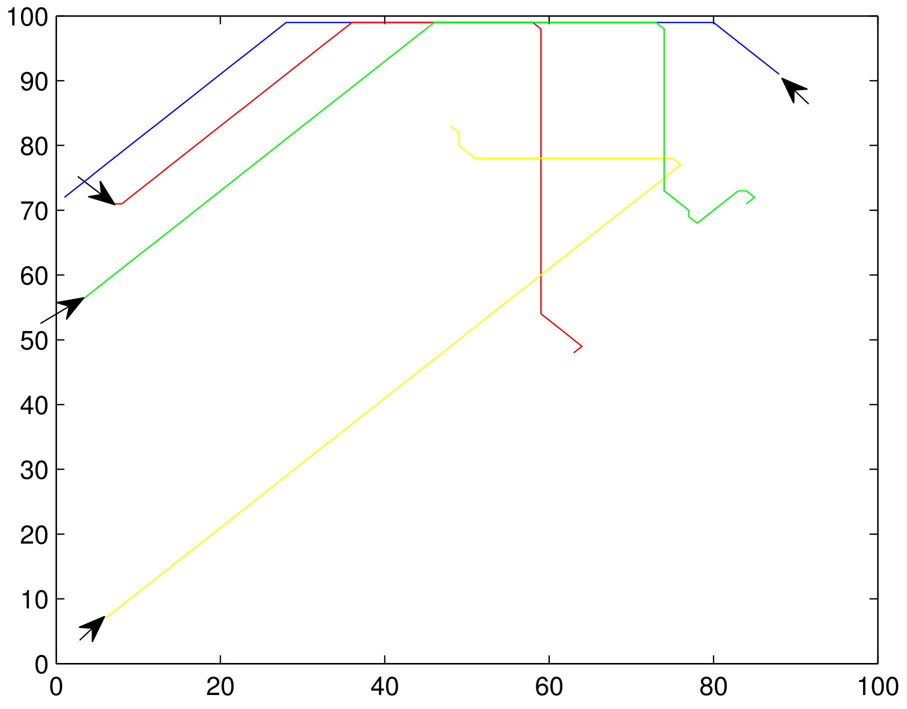


Fig. 4.10: Trajectories of UAVs in one simulation in which a crash occurs.

4.6 Comparison with Other Algorithms

In this chapter, we presented simulations with different numbers of UAVs and sizes of the field. Now, we want to compare our approach with other algorithms. In James' work [17], he compared the performances between three different algorithm: Mixed Integer Linear Programming (MILP), Artificial Potential Field (APF), and Dynamic Sparse A^* Search (DSAS). Here, we select our algorithm with highest survival rate to compare with James' result as shown in Fig. 4.11 and Fig. 4.12.

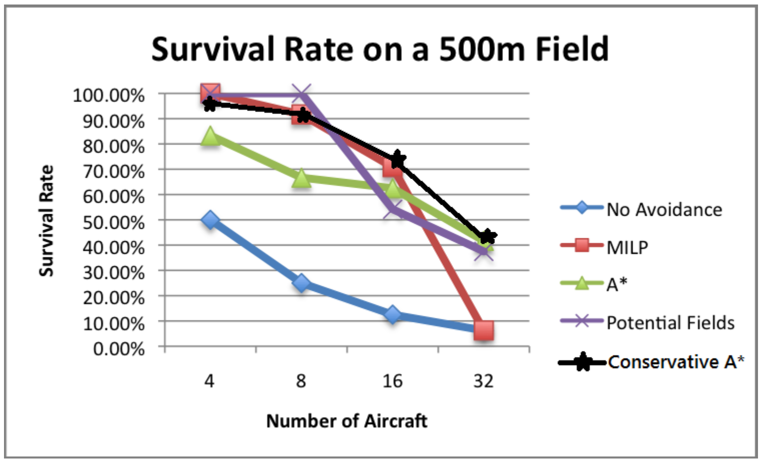


Fig. 4.11: Average survival rate in a 500 $m \times$ 500 m field [17]

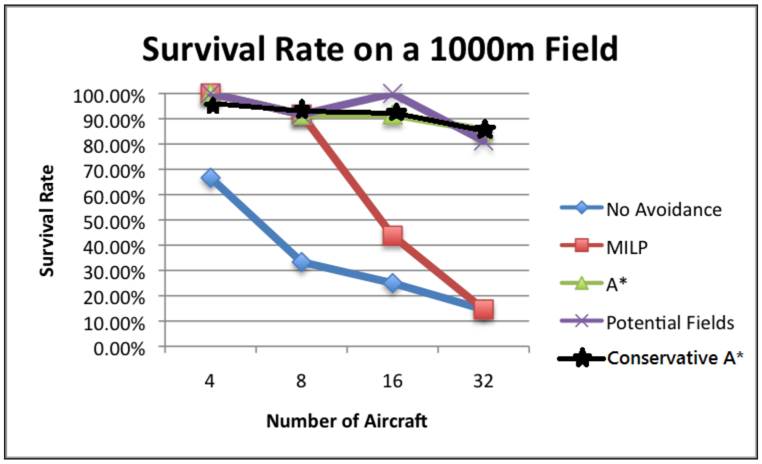


Fig. 4.12: Average survival rate in a 1000 $m \times$ 1000 m field [17]

Notes that the A^* in Fig. 4.11 and Fig. 4.12 is DSAS, and the Conservative A^* is the algorithm we use in this study. The survival rate of our approach can be calculated from Fig. 4.8 as shown in Fig. 4.13. Considering the survival rate, the conservative heuristic has a better result in most cases. In Comparison, for the scenarios with 16 UAVs in a 500 $m \times$500 m field, the survival rate of conservative heuristic is slightly higher than the MILP which is the best result in Fig. 4.11. For the scenarios in a 1000 $m \times$ 1000 m field, the survival rates of both aggressive and conservative heuristics are larger than 90% which is similar to the performance of the A^* in Fig. 4.12. Moreover, both MILP and APF have obvious strength and weakness. For instance, the APF has the highest survival rate in the scenarios on a field with lower density. However, the survival rate drops greatly in the scenario on a 500 $m \times 500$ m field when the number of UAVs is larger than 16. Therefore, even though our approach does not have 100% survival rate in most cases, it is still competitive when number of UAVs is smaller than 16 because it provides stably good survival rate under various configurations.

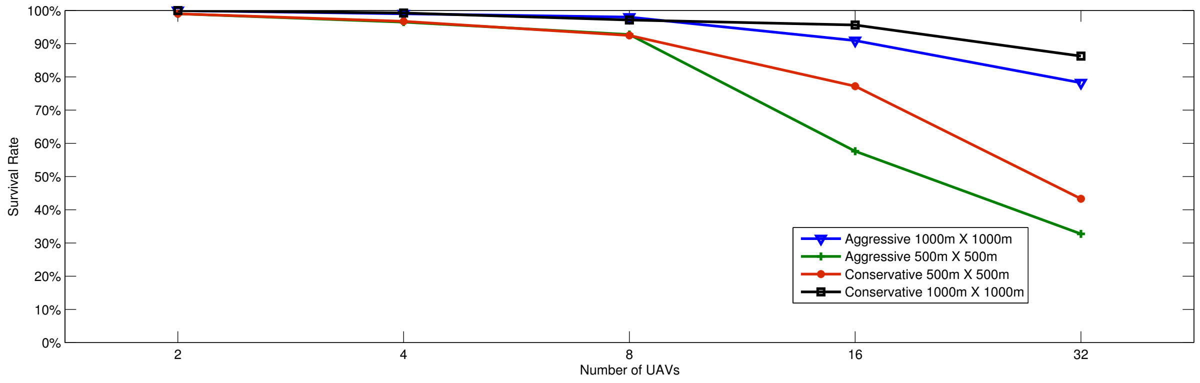


Fig. 4.13: Average survival rate of our approach

Chapter 5

Conclusion

In order to enable UAVs to fly autonomously, a reliable collision avoidance system is necessary. In Chapter 2, we reviewed several representative approaches to solve the UAV collision avoidance problem. Some of them have weaknesses such as computationally expensive algorithm [39], requirement of accurate sensor[24], requirement of additional equipment[38]. Then, we concluded that A^* is flexible by allowing different heuristic functions to handle different situations so that we can modify the original A^* to deal with dynamic obstacles. Therefore, A^* is a good candidate to perform the real time path planning.

Heuristics

Our main strategy of conservative heuristics is "do not pass through in front of the intruder". This strategy can at least avoid the imminent approach of other intruders. If we consider more situations of conflicts, the heuristics can possibly be improved. For example, we can consider the situation of parallel flying, instead of solving this situation, we can try to assign additional costs to prevent the occurrence of parallel flying. However, every time we add a new heuristic, we have to adjust the proportion of cost between path length (*Manhattan* method and safety (conservative or aggressive heuristics). In order to keep the safety and path length in an acceptable range, we choose a suitable proportion which is about $5:1$ for conservative heuristics and about $1:1$ for aggressive heuristics.

Future Works

According to the result of our study, we believe using two alternative heuristic functions in the $A*$ algorithm could achieve the best performance. Since aggressive heuristics work well when the number of UAVs is smaller than 8, we should use it for lower density condition. Once the number of UAVs is larger than 8, we can switch it to conservative heuristics to ensure the safety. We can also add more heuristics to consider a more complicated scenarios. However, as we mentioned above, we should be aware of the proportion between different heuristics while adding a new one.

Benefits to Robotics

The simulation result shows that the collision avoidance using $A*$ is reliable when the density(number of UAVs over the area of the field) is less than 8 UAVs/1,000,000 m^2. The relation between density and the performance of $A*$ may not be linear because flying more UAVs at the same time requires more complicated computations for their paths. This study investigated the application of $A*$ algorithm for path planning in a dynamic environment, and this is a common issue for mobile robots. Therefore, our study of $A*$ algorithm can be usefully adapted for other mobile robots.

References

[1] Online copies of SRI's proposals for the automaton project, subsequent progress reports, and related papers can be found at `http://www.ai.sri.com/shakey/`.

[2] Main site of Arduino team `http://arduino.cc/en/`.

[3] Introduction to TCAS II Version 7.1 `http://www.faa.gov/documentLibrary/media/Advisory_Circular/TCAS%20II%20V7.1%20Intro%20booklet.pdf`,.

[4] Final report of rtca task force 3, free flight implementation, November 1994.

[5] ICAO Annex 2. Rules of the air, 2006.

[6] Philippe Ballard and François Vacherand. The manhattan method: A fast cartesian elevation map reconstruction from range data. In *ICRA (3)*, pages 143–148, 1993.

[7] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[8] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[9] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, March 1999.

[10] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[11] J. E. Doran and D. Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 294(1437):pp. 235–259, 1966.

[12] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79:497–516, 1957.

[13] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[14] Veysel Gazi and Kevin M. Passino. Stability analysis of swarms. *IEEE Transactions on Automatic Control*, 48:692–697, 2003.

[15] Christopher M Geyer, Sanjiv Singh, and Lyle J. Chamberlain. Avoiding collisions between aircraft: State of the art and requirements for uavs operating in civilian airspace. Technical Report CMU-RI-TR-08-03, Robotics Institute, Pittsburgh, PA, March 2008.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths, 1968.

[17] James Holt. Comparison of aerial collision avoidance algorithms in a simulated environment, 2012.

[18] R.A. Howard. *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology, 1960.

[19] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *ARTIFICIAL INTELLIGENCE*, 101:99–134, 1998.

[20] L.V. Kantorovich. A new method of solving some classes of extremal problems, 1940.

[21] III Kelly, W.E. Conflict detection and alerting for separation assurance systems. In *Digital Avionics Systems Conference, 1999. Proceedings. 18th*, volume 2, pages 6.D.1–1 –6.D.1–8 vol.2, 1999.

[22] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, Spring 1986.

[23] J. Krozel and M. Peters. *Conflict Detection and Resolution for Free Flight*. 1997.

[24] J. Krozel and M. Peters. Strategic conflict detection and resolution for free flight. In *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, volume 2, pages 1822 –1828 vol.2, dec 1997.

[25] W.H. Kwon, A.M. Bruckstein, and T. Kailath. Stabilizing state-feedback design via the moving horizon method, 1983.

[26] N.E. Leonard and E. Fiorelli. Virtual leaders, artificial potentials and coordinated control of groups. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 3, pages 2968 –2973 vol.3, 2001.

[27] Patrick Lester. A* pathfinding for beginners. `http://www.policyalmanac.org/games/aStarTutorial.htm`, 2005. This is an electronic document. Date of publication: [Date unavailable]. Date retrieved: July 22, 2011. Date last modified: July 18, 2005.

[28] Michael L. Littman, Thomas L. Dean, and Leslie P. Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 394–402, Montreal, Québec, Canada, 1995.

[29] James Macgill. A* demonstration. `http://www.vision.ee.ethz.ch/~cvcourse/astar/AStar.html`, 1999. This is an electronic document. Date of publication: [Date unavailable]. Date retrieved: Oct 11, 2011. Last Updated January 18 1999.

[30] J. S. Morrel. The mathematics of collision avoidance in the air. journal of navigation. 11:18–28, 1958.

[31] N.J. Nilsson. *The quest for artificial intelligence: a history of ideas and achievements*. Cambridge University Press, 2010.

[32] J W Park, H D Oh, and M J Tahk. Uav collision avoidance based on geometric approach. *2008 Proceedings of Sice Annual Conference Vols 17*, pages 2039–2043 3382, 2008.

[33] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. Wiley series in probability and mathematical statistics: Applied probability and statistics. John Wiley & Sons, 1994.

[34] A. Richards and J.P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of American Control Conference*, volume 3, pages 1936–1941, 2002.

[35] N.D. Richards, M. Sharma, and D.G. Ward. A hybrid a*/automation approach to on-line path planning with obstacle avoidance, 2004.

[36] C. A. Rosen and N. J. Nilsson. Application of intelligent automata to reconnaissance. Technical report, Stanford Research Institute, November 1966. Project 5953 Interim Report 1 From the Nilsson archives SHAKEY papers.

[37] Tom Schouwenaars, Bart DeMoor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *In European Control Conference 2001*, pages 2603–2608, 2001.

[38] Karin Sigurd and Jonathan How. Uav trajectory design using total field collision avoidance, 2003.

[39] Selim Temizer, Mykel J. Kochenderfer, Leslie P. Kaelbling, Tomas Lozano-Pérez, and James K. Kuchar. Collision avoidance for unmanned aircraft using Markov decision processes. In *AIAA Guidance, Navigation, and Control Conference*, Toronto, Canada, 2010.

[40] J. van Tooren, M. Heni, A. Knoll, and J. Beck. Development of an autonomous avoidance algorithm for uavs in general airspace, 2007.

[41] Dalong Wang, Dikai Liu, and Gamini Dissanayake. A variable speed force field method for multi-robot collaboration. In *IROS*, pages 2697–2702, 2006.

[42] H. Paul Williams and Sally C. Brailsford. *Computational logic and integer programming*, pages 249–281. Oxford University Press, Inc., New York, NY, USA, 1996.